# Introduction to Scientific Python

CCNSS 2016

# Overview

# Overview

Basic Python knowledge

# Overview

Basic Python knowledge

LIF neuron simulation

# Overview

Basic Python knowledge

LIF neuron simulation

Structure your Python code

# Overview

Basic Python knowledge

LIF neuron simulation

Structure your Python code

Intro to scientific packages

# Overview

| | |
|---|---|
| Basic Python knowledge | **Part I** |
| LIF neuron simulation | |
| Structure your Python code | **Part II** |
| Intro to scientific packages | |

# Organization

Basic Python knowledge

**Part I**

LIF neuron simulation

Structure your Python code

**Part II**

Intro to scientific packages

# Flow

High-level slides

# Flow

High-level slides

In-depth technical notebooks

# Flow

High-level slides

In-depth technical notebooks

Exercise notebooks

# Flow

High-level slides

In-depth technical notebooks

Exercise notebooks ← **You code here!**

# Part I

# Topics Part I

Why Python?

# Topics Part I

Why Python?

Course Requirements

# Topics Part I

Why Python?

Course Requirements

Hello World

# Topics Part I

Why Python?

Course Requirements

Hello World

Variables

# Topics Part I

Why Python?

Course Requirements

Hello World

Variables

Control Flow

# Topics Part I

Why Python?

Course Requirements

Hello World

Variables

Control Flow

Plotting

# Why Python?

# Why Python?

Scientific computing for **FREE!**

# Why Python?

Scientific computing for **FREE!**

Easy to learn

# Why Python?

Scientific computing for **FREE!**

Easy to learn

Easy to read, maintain and extend

# Why Python?

Scientific computing for **FREE!**

Easy to learn

Easy to read, maintain and extend

Transferable programming skills

# Why Python?

Scientific computing for **FREE!**

Easy to learn

Easy to read, maintain and extend

Transferable programming skills

Powerful standard libraries

# Why Python?

Scientific computing for **FREE!**

Easy to learn

Easy to read, maintain and extend

Transferable programming skills

Powerful standard libraries

Interactive Mode

# Why Python?

Scientific computing for **FREE!**

Easy to learn

Easy to read, maintain and extend

Transferable programming skills

Powerful standard libraries

Interactive Mode

Portable, cross-platform

# Why Python?

Scientific computing for **FREE!**

Easy to learn

Easy to read, maintain and extend

Transferable programming skills

Powerful standard libraries

Interactive Mode

Portable, cross-platform

Highly scalable

# Popular Scientific Libraries

Numpy

Pandas

Scipy

Sympy

Matplotlib

IPython

# Popular Python Neuroscience Libraries

Brian

PyNN

NeuroTools

# Course Requirements

# Course Requirements

Anaconda Scientific Python Distribution

http://www.continuum.io/

BRIAN Spiking neural network simulator

http://briansimulator.org/

# Course Requirements

Anaconda Scientific Python Distribution **(v2.7)**

http://www.continuum.io/

BRIAN Spiking neural network simulator **(v2)**

http://briansimulator.org/

# Course Requirements

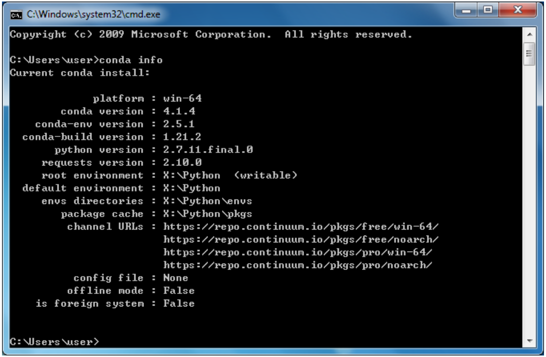
Check Anaconda installation by typing:

```
conda info
```

# Course Requirements

Check Anaconda installation by typing:

`conda info`

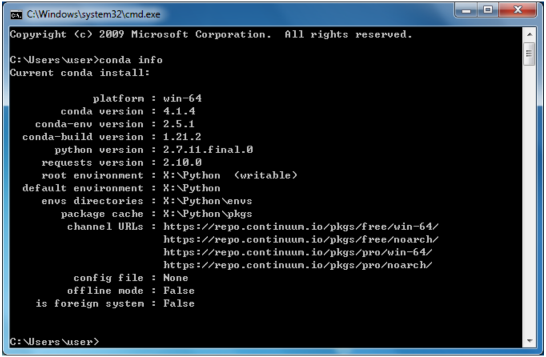


```
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\user>conda info
Current conda install:

             platform : win-64
        conda version : 4.1.4
    conda-env version : 2.5.1
  conda-build version : 1.21.2
       python version : 2.7.11.final.0
     requests version : 2.10.0
     root environment : X:\Python  (writable)
  default environment : X:\Python
     envs directories : X:\Python\envs
        package cache : X:\Python\pkgs
         channel URLs : https://repo.continuum.io/pkgs/free/win-64/
                        https://repo.continuum.io/pkgs/free/noarch/
                        https://repo.continuum.io/pkgs/pro/win-64/
                        https://repo.continuum.io/pkgs/pro/noarch/
          config file : None
         offline mode : False
    is foreign system : False


C:\Users\user>
```

# Course Requirements

Check Anaconda installation by typing:

`conda info`



```
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\user>conda info
Current conda install:

             platform : win-64
        conda version : 4.1.4
    conda-env version : 2.5.1
  conda-build version : 1.21.2
       python version : 2.7.11.final.0
     requests version : 2.10.0
     root environment : X:\Python  (writable)
  default environment : X:\Python
     envs directories : X:\Python\envs
        package cache : X:\Python\pkgs
         channel URLs : https://repo.continuum.io/pkgs/free/win-64/
                        https://repo.continuum.io/pkgs/free/noarch/
                        https://repo.continuum.io/pkgs/pro/win-64/
                        https://repo.continuum.io/pkgs/pro/noarch/
          config file : None
         offline mode : False
    is foreign system : False


C:\Users\user>
```

⚠ Switch to lab computer if error!

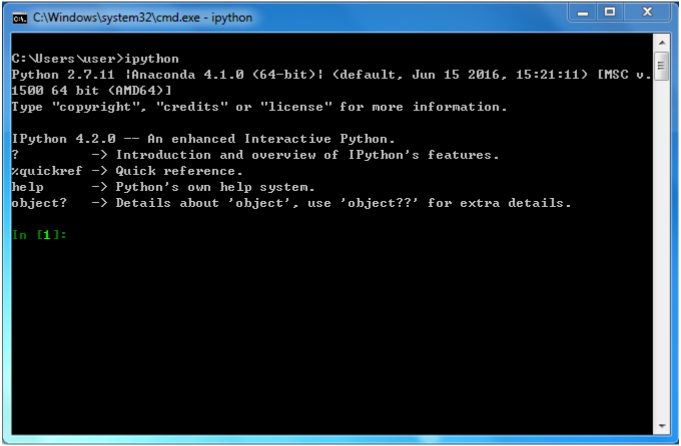
# Hello World

# Hello World - IPython

Start IPython shell by typing: `ipython`

# Hello World - IPython

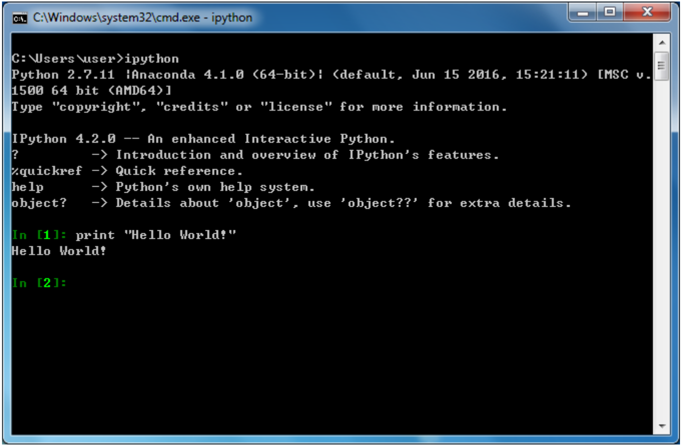
Start IPython shell by typing: `ipython`



```
C:\Users\user>ipython
Python 2.7.11 |Anaconda 4.1.0 (64-bit)| (default, Jun 15 2016, 15:21:11) [MSC v.
1500 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 4.2.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]:
```

# Hello World - IPython

Type `print "Hello World!"`



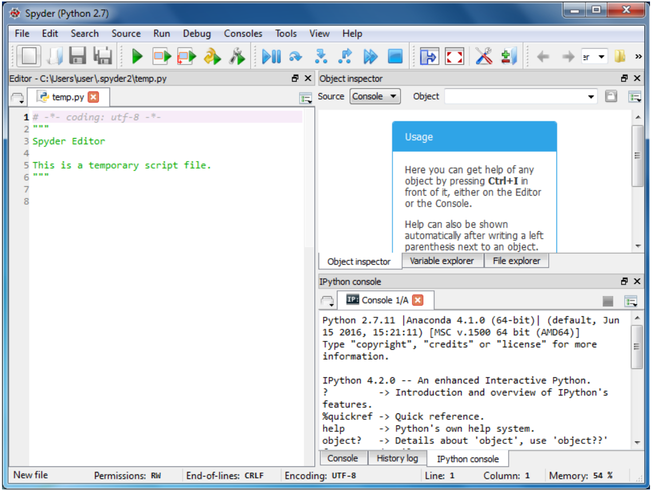
```
C:\Users\user>ipython
Python 2.7.11 |Anaconda 4.1.0 (64-bit)| (default, Jun 15 2016, 15:21:11) [MSC v.
1500 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 4.2.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: print "Hello World!"
Hello World!

In [2]:
```

# Hello World - Spyder

Start Spyder IDE by typing: `spyder`

# Hello World - Spyder

Start Spyder IDE by typing: `spyder`



```
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""
```

Usage

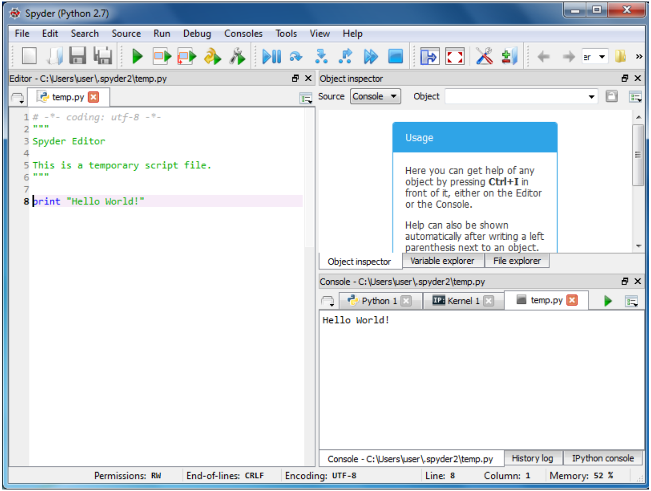
Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object.

```
Python 2.7.11 |Anaconda 4.1.0 (64-bit)| (default, Jun
15 2016, 15:21:11) [MSC v.1500 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more
information.

IPython 4.2.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's
features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??'
```

# Hello World - Spyder

Type `print "Hello World!"`



```
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""

print "Hello World!"
```

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object.

```
Hello World!
```

Permissions: RW  End-of-lines: CRLF  Encoding: UTF-8  Line: 8  Column: 1  Memory: 52 %

# Hello World - Jupyter Notebook

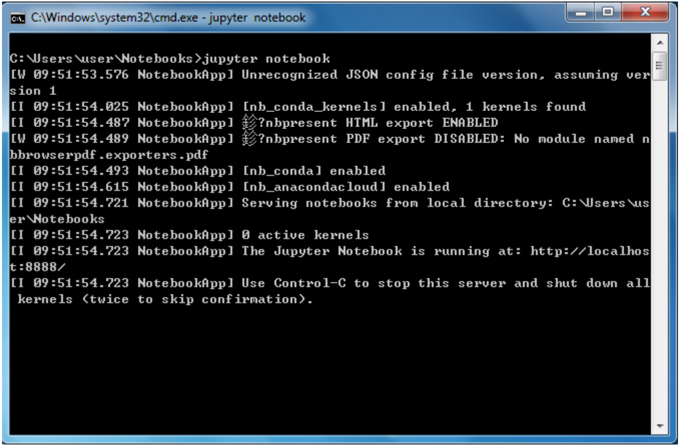
Start Jupyter Notebook by typing:

```
jupyter notebook
```

# Hello World - Jupyter Notebook

Start Jupyter Notebook by typing:

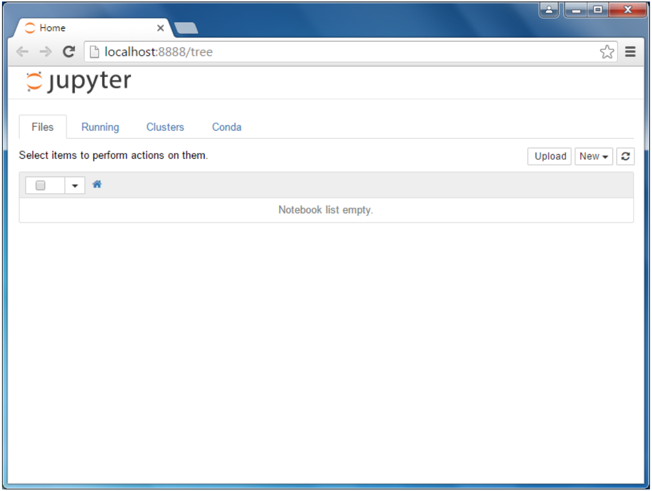
`jupyter notebook`



```
C:\Users\user\Notebooks>jupyter notebook
[W 09:51:53.576 NotebookApp] Unrecognized JSON config file version, assuming ver
sion 1
[I 09:51:54.025 NotebookApp] [nb_conda_kernels] enabled, 1 kernels found
[I 09:51:54.487 NotebookApp] 鉁?nbpresent HTML export ENABLED
[W 09:51:54.489 NotebookApp] 鉁?nbpresent PDF export DISABLED: No module named n
bbrowserpdf.exporters.pdf
[I 09:51:54.493 NotebookApp] [nb_conda] enabled
[I 09:51:54.615 NotebookApp] [nb_anacondacloud] enabled
[I 09:51:54.721 NotebookApp] Serving notebooks from local directory: C:\Users\us
er\Notebooks
[I 09:51:54.723 NotebookApp] 0 active kernels
[I 09:51:54.723 NotebookApp] The Jupyter Notebook is running at: http://localhos
t:8888/
[I 09:51:54.723 NotebookApp] Use Control-C to stop this server and shut down all
 kernels (twice to skip confirmation).
```

# Hello World - Jupyter Notebook

New web browser window → New→ Python

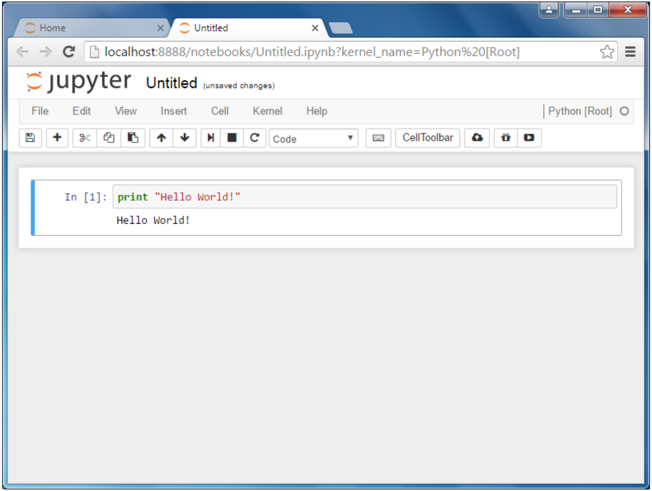


Select items to perform actions on them.

Notebook list empty.

# Hello World - Jupyter Notebook

Type `print "Hello World!"`



Home
Untitled
localhost:8888/notebooks/Untitled.ipynb?kernel_name=Python%20[Root]
jupyter
Untitled (unsaved changes)
File
Edit
View
Insert
Cell
Kernel
Help
Python [Root]
Code
CellToolbar

```
In [1]: print "Hello World!"

        Hello World!
```

# Using Links

`Start` → `All Programs` → `Anaconda2`

# Using Links

Start → All Programs → Anaconda2

→ IPython

→ Jupyter Notebook

→ Spyder

# Variables I

# Variable Names

### Valid names

Start with letter (`A-Z`, `a-z`) or underscore (`_`)

Followed by letters, underscore and digits `0-9`

# Variable Names

## Valid names

Start with letter (`A-Z`, `a-z`) or underscore (`_`)

Followed by letters, underscore and digits `0-9`

## Case sensitive

`neuron` $\neq$ `Neuron`

# Variable Names

## Valid names

Start with letter (`A-Z`, `a-z`) or underscore (`_`)

Followed by letters, underscore and digits `0-9`

## Case sensitive

`neuron` ≠ `Neuron`

## Reserved names

`import`, `lambda`, `If`, `True`...

# Standard Variable Types - Numbers

Integer (`int` and `long`)

```
a = 1
b = 314159265358979323846264338327950288
```

# Standard Variable Types - Numbers

Integer (`int` and `long`)

```
a = 1
b = 314159265358979323846264338327950288
```

Real and Complex (`float` and `complex`)

```
a = 1.0
b = 1e-9
c = 0.5 + 0.5j
```

# Operations with Numbers

| | | | | |
|---|---|---|---|---|
| Power | `**` | `5 ** 2` | → | 25 |

# Operations with Numbers

| | | | | |
|---|---|---|---|---|
| Power | `**` | `5 ** 2` | → | 25 |
| Multiplication | `*` | `2 * 3` | → | 6 |

# Operations with Numbers

| | | | | |
|---|---|---|---|---|
| Power | `**` | `5 ** 2` | → | 25 |
| Multiplication | `*` | `2 * 3` | → | 6 |
| Division | `/` | `14 / 3` | → | 4 |

# Operations with Numbers

| | | | | |
|---|---|---|---|---|
| Power | `**` | `5 ** 2` | → | 25 |
| Multiplication | `*` | `2 * 3` | → | 6 |
| Division | `/` | `14 / 3` | → | 4 |
| Modulo | `%` | `14 % 3` | → | 2 |

# Operations with Numbers

| | | | | |
|---|---|---|---|---|
| Power | `**` | `5 ** 2` | → | 25 |
| Multiplication | `*` | `2 * 3` | → | 6 |
| Division | `/` | `14 / 3` | → | 4 |
| Modulo | `%` | `14 % 3` | → | 2 |
| Addition | `+` | `1 + 2` | → | 3 |

# Operations with Numbers

| | | | | |
|---|---|---|---|---|
| Power | `**` | `5 ** 2` | → | 25 |
| Multiplication | `*` | `2 * 3` | → | 6 |
| Division | `/` | `14 / 3` | → | 4 |
| Modulo | `%` | `14 % 3` | → | 2 |
| Addition | `+` | `1 + 2` | → | 3 |
| Subtraction | `-` | `4 - 3` | → | 1 |

# Implicit Casting

From `int` to `float`

```
print "14 * 3 = ", 14 * 3
```

# Implicit Casting

From `int` to `float`

```
print "14 * 3 = ", 14 * 3
```

```
14 * 3 = 42
```

# Implicit Casting

From `int` to `float`

```
print "14 * 3 = ", 14 * 3
```

```
14 * 3 = 42
```

```
print "14 * 3.0 =", 14 * 3.0
```

# Implicit Casting

From `int` to `float`

```
print "14 * 3 = ", 14 * 3
```

```
14 * 3 = 42
```

```
print "14 * 3.0 =", 14 * 3.0
```

```
14 * 3.0 = 42.0
```

# Variable Types - Strings

Strings (`str`)

```
a = ’Hello World!’
b = "Mixin’ quotes"
print b
```

# Variable Types - Strings

Strings (`str`)

```
a = 'Hello World!'
b = "Mixin' quotes"
print b
```

```
Mixin' quotes
```

# Variable Types - Boolean

Boolean (`bool`)

```
c = True
print c
```

# Variable Types - Boolean

Boolean (`bool`)

```
c = True
print c
```

```
True
```

# Variable Assignment

## Individual assignment

```
counter = 100

price = 1000.0

name = "John"
```

# Variable Assignment

## Individual assignment

```
counter = 100

price = 1000.0

name = "John"
```

## Multiple assignment

```
counter, price, name = 100, 1000.0, "John"

a1 = a2 = a3 = 1
```

# Variable Assignment

## Individual assignment

```
counter = 100
price = 1000.0
name = "John"
```

## Multiple assignment

```
counter, price, name = 100, 1000.0, "John"
a1 = a2 = a3 = 1
```

## Assignment with operation (`+ - * / % ** //`)

```
price += 100
```

# Standard Variable Types - Lists

```
mylist = [100, 1000.0, "John", 0.5 + 0.5j]

print mylist
```

# Standard Variable Types - Lists

```
mylist = [100, 1000.0, "John", 0.5 + 0.5j]
print mylist
```

```
[100, 1000.0, ’John’, (0.5+0.5j)]
```

# Standard Variable Types - Lists

```
mylist = [100, 1000.0, "John", 0.5 + 0.5j]
print mylist
```

```
[100, 1000.0, 'John', (0.5+0.5j)]
```

```
mylist = mylist + [10.0]
print mylist
```

# Standard Variable Types - Lists

```
mylist = [100, 1000.0, "John", 0.5 + 0.5j]
print mylist
```

```
[100, 1000.0, ’John’, (0.5+0.5j)]
```

```
mylist = mylist + [10.0]
print mylist
```

```
[100, 1000.0, ’John’, (0.5+0.5j), 10.0]
```

# Standard Variable Types - Lists

```
mylist = [100, 1000.0, "John", 0.5 + 0.5j]
print mylist
```

```
[100, 1000.0, ’John’, (0.5+0.5j)]
```

```
mylist = mylist + [10.0]
print mylist
```

```
[100, 1000.0, ’John’, (0.5+0.5j), 10.0]
```

```
mylist = ["Howdy!"] + mylist
print mylist
```

# Standard Variable Types - Lists

```
mylist = [100, 1000.0, "John", 0.5 + 0.5j]
print mylist
```

```
[100, 1000.0, ’John’, (0.5+0.5j)]
```

```
mylist = mylist + [10.0]
print mylist
```

```
[100, 1000.0, ’John’, (0.5+0.5j), 10.0]
```

```
mylist = ["Howdy!"] + mylist
print mylist
```

```
[’Howdy!’, 100, 1000.0, ’John’, (0.5+0.5j), 10.0]
```

# Comments & Line Breaks

## Comments

```
a = 1.0
print a # this is a comment
```

# Comments & Line Breaks

## Comments

```
a = 1.0
print a # this is a comment
```

```
1.0
```

# Comments & Line Breaks

## Comments

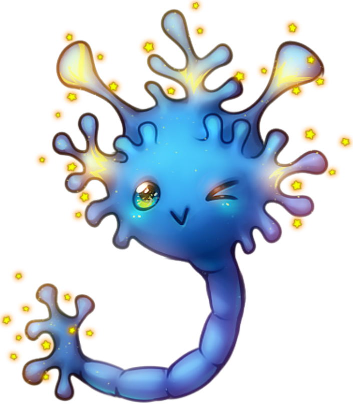
```
a = 1.0
print a # this is a comment
```

```
1.0
```

## Line breaks

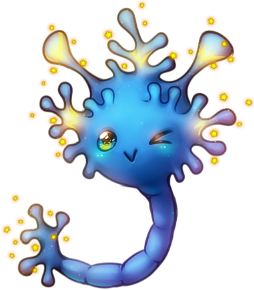
```
b =  1 + 2 + 3 + 4 + 5 + 6 + 7 \
     + 8 + 9 + 10
```

# Basic Variables
# Notebook



# Coding Time!

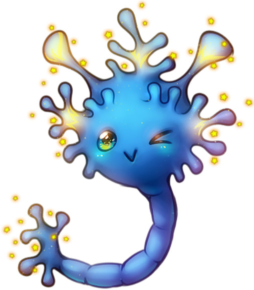
# LIF Neuron Exercise



### Objective

Implement LIF neuron

# LIF Neuron Exercise

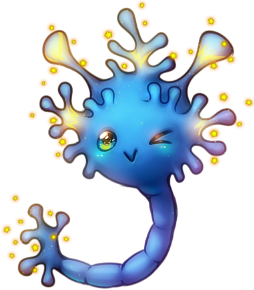


## Objective

Implement LIF neuron

Extract ensemble stats

# LIF Neuron Exercise



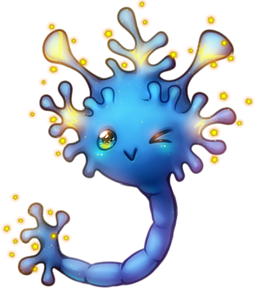
## Objective

Implement LIF neuron

Extract ensemble stats

Produce nice graphs!!!

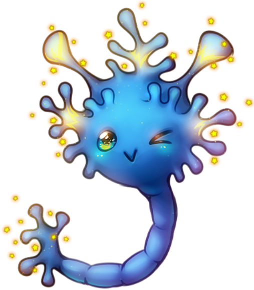
# LIF Neuron Exercise



## Strategy

No spikes first

# LIF Neuron Exercise

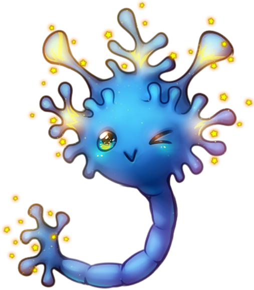


### Strategy

No spikes first

Implement ODE integration

# LIF Neuron Exercise



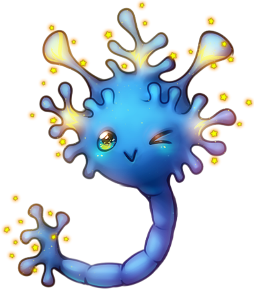
### Strategy

No spikes first

Implement ODE integration

Extend to ensemble stats

# LIF Neuron Exercise



### Strategy

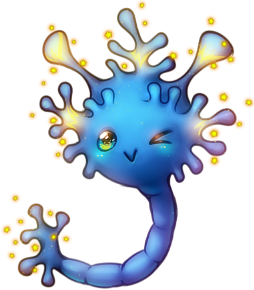
No spikes first

Implement ODE integration

Extend to ensemble stats

Validate stats $\rightleftharpoons$ white noise input

# LIF Neuron Exercise



## Strategy

No spikes first

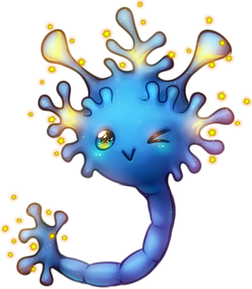
Implement ODE integration

Extend to ensemble stats

Validate stats $\rightleftharpoons$ white noise input

Introduce spikes

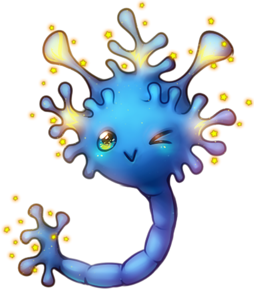
# LIF Neuron Exercise



**Coding Time!**

Start IPython Notebook

# LIF Neuron Exercise



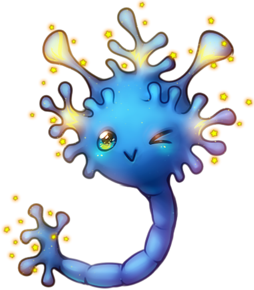
**Coding Time!**

Start IPython Notebook

(Exercise 1)

Encode simulation parameters

# LIF Neuron Exercise



## Simulation parameters

```
t_max = 0.1         # second
dt = 1e-3           # second
tau = 20e-3         # second
el = -60e-3         # volt
vr = -70e-3         # volt
vth = -50e-3        # volt
i_mean = 25e-3      # ampere
```

# Control Flow

# Control Flow - Loops

## While loop

```
t, t_max, dt = 0, 10, 1

while t < t_max:

    print t

    t += dt

print "Finished at value t = ", t
```

# Control Flow - Loops

## While loop

```
t, t_max, dt = 0, 10, 1

while t < t_max:

    print t

    t += dt

print "Finished at value t = ", t
```

```
0
⋮
9
Finished at value t = 10
```

# Control Flow - Loops

## For loop

```
for t in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]:
    print t
print "Finished at value t = ", t
```

# Control Flow - Loops

## For loop

```
for t in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]:
    print t
print "Finished at value t = ", t
```

```
0
⋮
9
Finished at value t = 9
```

# Control Flow - Loops

## For loop

```
t_max, dt = 10, 1

for t in range(0, t_max, dt):
    print t

print "Finished at value t = ", t
```

# Control Flow - Loops

## For loop

```
t_max, dt = 10, 1

for t in range(0, t_max, dt):
    print t

print "Finished at value t = ", t
```

```
0
⋮
9
Finished at value t = 9
```

# Indentation

Indentation = logical structure

# Indentation

Indentation = logical structure

Same spacing = same logical block

# Indentation

Indentation = logical structure

Same spacing = same logical block

Use 4 whitespaces (PEP 8)

http://legacy.python.org/dev/peps/pep-0008/

# Indentation

```
for t in range(0, t_max, dt):
␣␣␣␣print t
```

# Control Flow - Conditional

## If statement

```
t_max = 10

if t_max >= 5:
    print "t_max is equal to or more than 5 s"
```

# Control Flow - Conditional

## If statement

```
t_max = 10

if t_max >= 5:
    print "t_max is equal to or more than 5 s"
```

```
t_max is equal to or more than 5 s
```

# Control Flow - Conditional

## If-Else statements

```
t_max = 10

if t_max < 5:
    print "t_max is less than 5 s"
else:
    print "t_max is equal to or more than 5 s"
```

# Control Flow - Conditional

## If-Else statements

```
t_max = 10

if t_max < 5:
    print "t_max is less than 5 s"
else:
    print "t_max is equal to or more than 5 s"
```

```
t_max is equal to or more than 5 s
```

# Control Flow - Conditional

## If-Elif-Else statements

```
t_max = 10

if t_max < 1:
    print "t_max is less than 1 s"
elif t_max <= 0.5:
    print "t_max is between 1 and 5 s"
else:
    print "t_max is more than 5 s"
```

# Control Flow - Conditional

## If-Elif-Else statements

```
t_max = 10

if t_max < 1:
    print "t_max is less than 1 s"
elif t_max <= 0.5:
    print "t_max is between 1 and 5 s"
else:
    print "t_max is more than 5 s"
```

```
t_max is more than 5 s
```

# Break & Continue

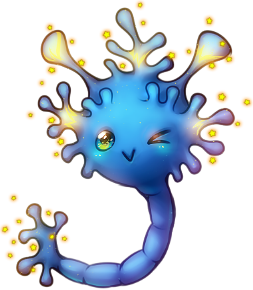
## Break and Continue statements

```
t, t_max, dt = 0, 10, 1

while t <= t_max:
    if t > 5:
        print "I’m done!"
        break
    elif t % 2 == 0:
        print t, "is even"
        t += dt
        continue
    t += dt

print "Finished at value t = ", t
```

# Control Flow

# Notebook

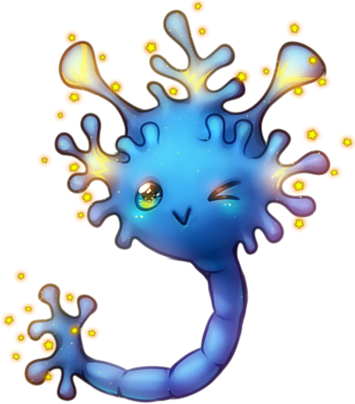
# LIF Neuron Exercise



Membrane equation

$$\tau_m \frac{d}{dt} V(t) = E_L - V(t) + RI(t)$$

# LIF Neuron Exercise



**Coding Time!**

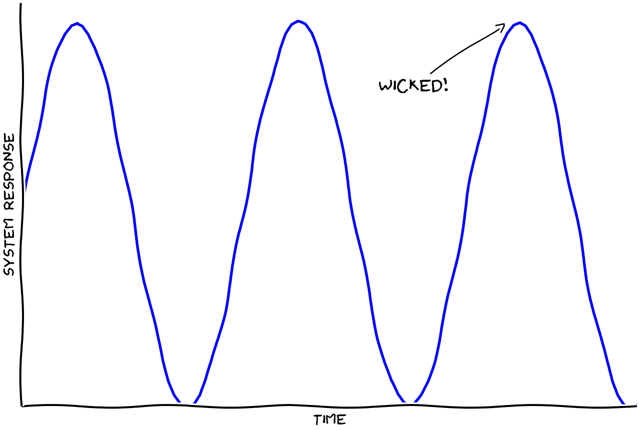
(Exercise 2)

Discrete time integration of $V(t)$

$$V(t+\Delta t) = V(t) + \Delta t(\cdots)$$

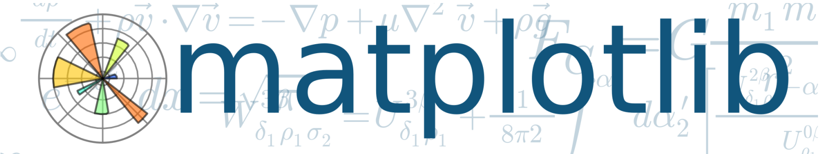
# Plotting

# Showing Your Stuff



SOME OSCILLATORY SYSTEM

# Matplotlib Library



# Simple Plot

Key function:

```
plot(x, y, 'r+', label='cross')
```

# Simple Plot

Key function:

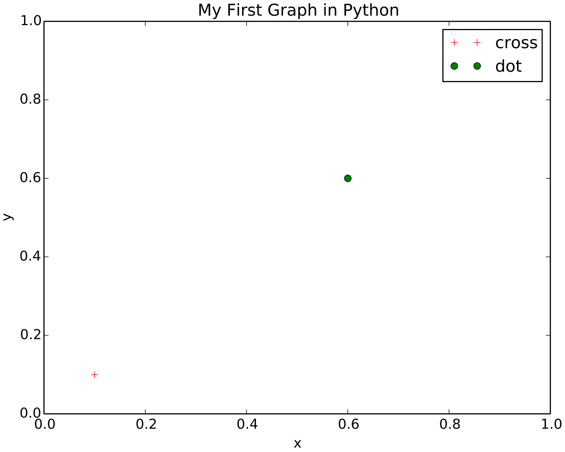
```
plot(x, y, 'r+', label='cross')
```

will plot a red cross at position $(x, y)$ with label 'cross'

# Simple Plot

```
import matplotlib.pyplot as plt

x1, y1, x2, y2 = 0.1, 0.1, 0.6, 0.6

plt.figure()
plt.plot(x1, y1, ’r+’, label=’cross’)
plt.plot(x2, y2, ’go’, label=’dot’)

plt.title(’My First Graph in Python’)
plt.xlabel(’x’)
plt.ylabel(’y’)
plt.legend()

plt.show()
```

# Simple Plot


My First Graph in Python
cross
dot
y
x
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

# Plotting Lists

```
x = range(10)

print x
```

# Plotting Lists

```
x = range(10)
print x
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

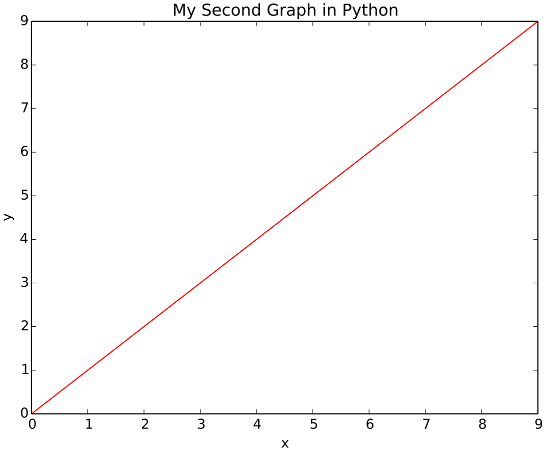
# Plotting Lists

```
x = range(10)
print x
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

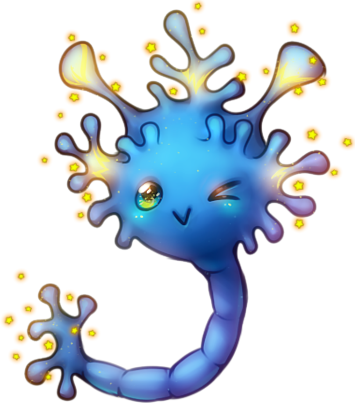
```
plot(x, x, ’ro’)
```

# Simple Plot II



# Plotting Notebook

# LIF Neuron Exercise



**Coding Time!**

(Exercise 3)

Plot $V(t)$ time course

(Exercise 4)

Stochastic input currents

# Variables II

# List Indexing

```
mylist = [100, 1000.0, "John", 0.5 + 0.5j]
print mylist[0]
```

# List Indexing

```
mylist = [100, 1000.0, "John", 0.5 + 0.5j]
print mylist[0]
```

```
100
```

# List Indexing

```
mylist = [100, 1000.0, "John", 0.5 + 0.5j]
print mylist[0]
```

```
100
```

```
mylist = [100, 1000.0, "John", (0.5+0.5j), 10.0]
del mylist[-1]
print mylist
```

# List Indexing

```
mylist = [100, 1000.0, "John", 0.5 + 0.5j]
print mylist[0]
```

```
100
```

```
mylist = [100, 1000.0, "John", (0.5+0.5j), 10.0]
del mylist[-1]
print mylist
```

```
[100, 1000.0, 'John', (0.5+0.5j)]
```

# List Slicing

```
mylist = [100, 1000.0, "John", 0.5 + 0.5j]
print mylist[1:3]
```

# List Slicing

```
mylist = [100, 1000.0, "John", 0.5 + 0.5j]
print mylist[1:3]
```

```
[1000.0, ’John’]
```

# List Slicing

```
mylist = [100, 1000.0, "John", 0.5 + 0.5j]
print mylist[1:3]
```

```
[1000.0, ’John’]
```

```
print mylist[1:]
```

# List Slicing

```
mylist = [100, 1000.0, "John", 0.5 + 0.5j]
print mylist[1:3]
```

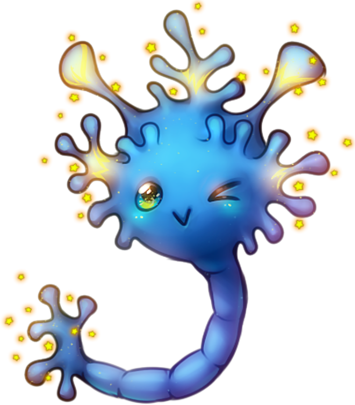
```
[1000.0, ’John’]
```

```
print mylist[1:]
```

```
[1000.0, ’John’, 0.5 + 0.5j]
```

# Working with Lists
# Notebook

# LIF Neuron Exercise



**Coding Time!**

(Exercise 5, 6, 7 and 8)

Ensemble statistics

the sample standard variation

# Variables III

# Standard Variable Types - Dictionary

```
mydict = {'qty': 100, 'person': "John"}
print mydict
```

# Standard Variable Types - Dictionary

```
mydict = {’qty’: 100, ’person’: "John"}
print mydict
```

```
{’person’: ’John’, ’qty’: 100}
```

# Standard Variable Types - Dictionary

```
mydict = {’qty’:  100, ’person’:  "John"}
print mydict
```

```
{’person’:  ’John’, ’qty’:  100}
```

```
print mydict[’person’]
```

# Standard Variable Types - Dictionary

```
mydict = {’qty’:  100, ’person’:  "John"}
print mydict
```

```
{’person’:  ’John’, ’qty’:  100}
```

```
print mydict[’person’]
```

```
John
```

# Standard Variable Types - Dictionary

```
mydict = {’qty’:  100, ’person’:  "John"}
print mydict.keys()
```

# Standard Variable Types - Dictionary

```
mydict = {’qty’:  100, ’person’:  "John"}
print mydict.keys()
```

```
[’person’, ’qty’]
```

# Standard Variable Types - Dictionary

```
mydict = {’qty’:  100, ’person’:  "John"}
print mydict.keys()
```

```
[’person’, ’qty’]
```

```
print mydict.values()
```

# Standard Variable Types - Dictionary

```
mydict = {’qty’:  100, ’person’:  "John"}
print mydict.keys()
```

```
[’person’, ’qty’]
```

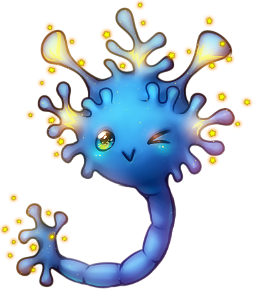
```
print mydict.values()
```

```
[’John’, 100]
```

# Dictionaries

# Notebook

# LIF Neuron Exercise



Membrane equation
with reset condition

If $V(t) < V_{th}$

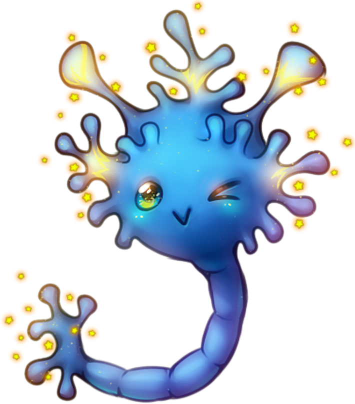
$$\tau_m \frac{d}{dt} V(t) = E_L - V(t) + RI(t)$$

Else

$$V(t) = V_r$$

record spike at time $t$

# LIF Neuron Exercise



**Coding Time!**

(Exercise 9)

Output spikes

(Exercise 10)

Refractory period
Integration step

# Recap

# Overview

| | |
|---|---|
| Basic Python knowledge | **Part I** |
| LIF neuron simulation | |
| Structure your Python code | **Part II** |
| Intro to scientific packages | |

# Overview

✓ Basic Python knowledge **Part I**

✓ LIF neuron simulation

Structure your Python code **Part II**

Intro to scientific packages

# End Part I